PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 34/07

Treating Literary Genres as Application Domains

Angelo E. M. Ciarlini, Marco A. Casanova, Antonio L. Furtado, Paulo. A.S. Veloso

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 34/07 ISSN 0103-9741
Editor: Prof. Carlos José Pereira de Lucena December 2007

Treating Literary Genres as Application Domains*

Angelo E. M. Ciarlini[1], Marco A. Casanova[2], Antonio L. Furtado[2], Paulo. A.S. Veloso[3]

[1]UniRio – Departamento de Informática Aplicada
[2]Pontifícia Universidade Católica do Rio de Janeiro – Departamento de Informática
[3]UFRJ – COPPE Sistemas
Rio de Janeiro - Brasil

Abstract: Literary genres, of prime relevance to storytelling, can be regarded as a particular kind of application domain. As such, they can be usefully characterized by combining notions drawn from literary theory with well-known models developed for information systems. Once a genre is specified with some rigor in a constructive way, it becomes possible to determine whether a given plot is a legitimate representative of the genre, as well as to generate such plots. This paper presents a conceptual modeling method with this purpose, based on a plan recognition / plan generation paradigm. The method leads to the formulation of static, dynamic and behavioral schemas, expressed in temporal logic, and allows multi-stage interactive plot generation. The paper also describes a prototype tool, developed to support the method, and includes a case study, involving a simple Swords and Dragons genre.

Keywords: Storytelling, Literary Genres, Application Domains, Conceptual Modeling, Simulation, Logic Programming.

Resumo: Gêneros literários, de extrema relevância no campo de narração de estórias, podem ser encarados como espécie particular de domínio de aplicação. Por conseguinte, é útil caracterizá-los através de uma combinação de noções extraídas da teoria literária com modelos originariamente criados para sistemas de informação. Após especificar um gênero, com suficiente rigor e de forma construtiva, torna-se possível determinar se um dado enredo é representante legítimo do gênero, bem como gerar tais enredos. Este trabalho apresenta um método de modelagem conceitual com este propósito, baseado em um paradigma de reconhecimento / geração de planos. O método conduz à formulação de esquemas estáticos, dinâmicos e comportamentais, expressos em lógica temporal, e permite a geração interativa de enredos através de estágios múltiplos. O trabalho também descreve um protótipo de ferramenta de suporte ao método, e inclui um estudo de caso envolvendo um gênero simples de Espadas e Dragões.

Palavras-chave: Narração de Estórias, Gêneros Literários, Domínios de Aplicação, Modelagem Conceitual, Simulação, Programação em Lógica.

* This work has been partly sponsored by the Ministério de Ciências e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

1. Introduction

In this paper, we address the question: What is a literary genre? Although it seems obvious that, in its full generality, the question will always remain outside the reach of a complete formal treatment, we claim that a useful approximation is attainable and proceed to introduce a specification method that captures aspects of an intended genre. In addition, we describe a prototype tool that generates story plots belonging to a (formalized) genre and determines whether a story plot can be classified as belonging to the genre.

Studies in narratology [3] suggest that the composition of stories is a three-layered process and that each layer can be analyzed separately: fabula, story and text. Informally speaking, a fabula is a series of events happening in a real or fictional world. A story corresponds to how a fabula is reported by the author. Finally, a text is a materialization of a story in some medium, such as text, motion picture, or animation.

To give an example, the mythical career of Ulysses, with the events in strict chronologic order, is the fabula that Homer had in mind when composing the *Odyssey*. Homer's story concentrates on the hero's homecoming and is organized in twenty-four books, in some of which the poet allows Ulysses to tell his own exploits in a long "flash-back" (technically, a case of anachrony). Finally, the epic text produced by Homer is a poem in dactylic hexameter verses composed in classic Greek. Clearly, a prose translation of the *Odyssey* in a modern language would be a different text, but, if it is a faithful rendering, it should preserve the story as narrated by Homer, and consequently the original fabula as well. A more radical change may occur when someone retells the story, cutting or summarizing a number of episodes, linearizing what remains so as to eliminate the anachronies, etc., which, of course, results in a different story. Extreme cases of change would even modify the fabula, for example, by eliminating as allegedly incompatible with modern taste the interventions of the goddess Athena, and combining in one character (a conflation) two or more of Penelope's suitors.

In our work, we shall deal exclusively with the fabula layer. Accordingly, we shall view a genre as a set of plots, taking the word *plot* in the sense of a partially ordered sequence of events. In addition, the types of events allowed in the genre being defined will be restricted to a fixed repertoire, as proposed by the Russian literary theoretician Vladimir Propp in his seminal work on the fairy-tales genre [31]. This decision classifies our approach as primarily plot-based, in the terminology of storytelling research [22, 36], as opposed to a character-based orientation [8]. However, it will become clear, in the course of the paper, that we also contemplate some character-based aspects.

The method we propose in this paper comprises three levels of conceptual modeling that provide: (a) a description of the mini-world wherein the narrative takes place; (b) what events can be enacted by the participants; and (c) what motives guide their behavior.

We combine several modeling notions, borrowed from literary theory [1,3,4,14,31,35] and from information systems. Plots belonging to a genre are, to a certain extent, comparable to sentences belonging to a language, which suggests the use of some Chomsky grammar, such as Rumelhart's story grammar [33], as a mechanism to accomplish purposes (a) and (b) above. However, we opted for a plan-recognition / plan-generation paradigm [21], which is fully compatible with the nature of the schemas and is particularly apt to cope with semantic and pragmatic aspects. The formalism is based on temporal logic [11]

and the notation adopts the clausal format required by the Prolog logic programming language, in which the planning algorithms were written.

In [19], we argued that temporal databases are, in general, repositories of narratives about the agents and objects involved. In [21], we showed that plan-generation (and plan-recognition) algorithms can be used in the context of database narratives for decision support. An operational description of the modeling issues of this scenario can be found in [20], while a characterization of the machinery involved in the process is given in [10,11]. Our formal framework can be compared to the event calculus [28], especially to one of its variants, the event calculus with preconditions [9]. The description of narratives with situation calculus [29] enables us to better reason about the hypothetical situations that result from hypothetical actions. The difference in our approach is that we focus on the creation of coherent narratives: we reason about the situations that hold or may hold along a narrative in order to infer goals that will bring about new actions, which will in turn conduct the narrative.

We exemplify the proposed method by modeling an elementary Swords and Dragons genre. In the example, princesses, knights, dragons and magicians play the roles of victims, heroes and villains. They perform actions such as attack, kidnap, fight, kill and marry. Our prototype tool was able to generate (and recognize) dozens of quite different plots, all of them fully compatible with the formal model. This simple example demonstrated that the method provides a solid background for interactive storytelling, since it guarantees the coherence of the stories. In addition, the generation of unexpected stories was useful to point out that we were implicitly assuming constraints that had not been formalized. In this way, the specification of a formal model proved to be useful to help us understand a genre. We refer the reader to [10] for the thesis of the first author, which initiated this project..

The paper[1] is organized as follows. Sections 2, 3 and 4 introduce the three levels of conceptual modeling we propose. Section 5 describes the application of the method and the use of the prototype for our elementary Swords and Dragons genre. Section 6 concludes the paper. The appendices contains the full Prolog code for the example.

2. Static schema

2.1 Informal description of the static schema

The efficacy of the *entity-relationship model* (ER model), with a number of extensions, has long been amply recognized in the realm of the application domains of business information systems [15]. We argue in what follows that it can be equally helpful for modeling the static aspects of literary genres.

Briefly, an *entity* is anything of interest by itself, material or abstract, animate or not. Entities form *classes*, whose *instances* are distinguished by the values of an *identifier*, which we assume to be a single attribute. In addition to the identifier, other *attributes* may characterize the entity instances. Attributes have *values* of some type (alphabetic, numerical, etc.). Attributes of type *Boolean* (with values `true` or `false`) and *composite* attributes (with sub-divisions) are special cases. Entity classes may be associated through *relationships*, which we assume to be binary.

[1] A version of the present text [12] has been submitted for publication.

The original ER model is typically extended to include the concept of class generalization / specialization, expressed through the (transitive) *is-a* relation [27,30,37]. Given two classes *C* and *C'*, if one indicates that *C is-a C'*, then we say that *C specializes C'* and that *C'* is *more general* than *C*. In this case, *C* inherits all attributes defined for *C'* and each instance of *C* is also an instance of *C'*.

One more addition to the ER model is convenient to help bridge the dynamic and behavioral levels, addressed in Sections 3 and 4. Whereas the other qualifying notions refer to what the entities *are*, in order to indicate how they are expected to *act*, we need to assign *roles* to certain entities (in the theatrical sense, and in the sense of the *agent* concept, used in Artificial Intelligence and Software Engineering).

We call a *fact* an assertion about the existence of an entity instance, the value of an attribute of an entity instance, the existence of a relationship instance, the value of an attribute of a relationship instance or the assignment of a role to an entity instance. The set of facts holding at a given point in time constitutes a *database state*.

The static specification of a genre, exactly as for a business application domain, requires that only *valid* states be admitted. A valid state must conform to certain *static integrity constraints*. Some constraints are native to the ER model: relationship instances can only involve existing entity instances, attributes are in general single-valued (even though they may change along time), attribute values must be of the specified type, etc. Other integrity constraints are imposed by *conventions* of the genre or by *regulations* of the application domain. They are expressed in some appropriate notation, outside the basic ER model.

Section 5.2 contains the complete static schema for our Swords and Dragons genre.

2.2 Formalization of the static schema

In this section, we introduce a formal framework to express static schemas in the ER model. The formalization follows Ciarlini and Furtado [11] and is based on the standard syntax and semantics of first-order languages. Therefore, we detail only the concepts that directly matter to our discussion.

A *plot language* is a first-order language, with equality, whose alphabet contains a set of *database symbols* partitioned into:

entity class names: unary predicate symbols to denote entity class names

Boolean entity attribute names: unary predicate symbols to denote Boolean entity attribute names

simple entity attribute names: binary predicate symbols to denote simple entity attribute names

composite entity attribute names: n-ary predicate symbols to denote composite entity attribute names

relationship names: binary predicate symbols to denote relationship class names (only binary relationships are considered)

Boolean relationship attribute names: binary predicate symbols to denote Boolean relationship attribute names

simple relationship attribute names: binary predicate symbols to denote simple relationship attribute names

composite relationship attribute names: n-ary predicate symbols to denote composite relationship attribute names

role names: unary predicate symbols to denote roles

database constants: constants to denote data values

Besides the database symbols, the alphabet of a plot language contains *constraint predicate symbols*, corresponding to constraint predicates over concrete domains (such as equalities and inequalities over the real numbers), and *constraint function symbols*, corresponding to functions over concrete domains (such as addition and subtraction over the real numbers).

To formalize the static schema of our Swords and Dragons genre, we introduce the following database symbols:

entity class names: `creature, person, knight, princess, magician, dragon, place`

Boolean entity attribute names: `alive`

simple entity attribute names: `name, nature, strength, place_name`

composite entity attribute name: `protection` (a ternary predicate symbol, indicating the kind and level of protection of a place)

relationship names: `home, current_place, acquaintance, married, kidnapped`

simple relationship attribute names: `affection`

role name: `hero, victim, villain, donor`

database constants: `'Marian', 'White_Palace', 'Hoel'`

Sample database facts are:

`princess('Marian')` (`'Marian'` is an instance of entity class `princess`)

`strength('Marian',10)` (the value of attribute `strength` for `'Marian'` is `10`)

`alive('Marian')` (the value of attribute `alive` for `'Marian'` is `True`)

`acquaintance('Marian','Hoel')`
(`'Marian'` and `'Hoel'` are instances related by the relationship `acquaintance`)

`affection('Marian','Hoel',0)`
(the value of attribute `affection` for `'Marian'` and `'Hoel'` is `0`)

`place('White_Palace')` (`'White_Palace'` is instance of entity class `place`)

`protection('White_Palace',1,70)`
(the value of the composite attribute `protection` for `'White_Palace'` is `1` and `70`, respectively indicating the kind and level of protection)

`victim('Marian')` (`'Marian'` plays the role of `victim`)

In what follows, let L be a plot language. The syntax and semantics of L follow as for first-order languages, so that we will concentrate just on the details that matter to our discussion.

A *substitution* is a function *i* mapping each variable of L into a term and a *ground substitution* is a substitution mapping each variable into a variable-free term. An *expression* is a formula or a term. Given an expression *e*, we use *e[i]* to denote the *substituted version* of *e*, obtained by replacing each free variable in *e* in accordance with *i*.

A *database literal* is an atomic formula with a database predicate symbol. A *ground database literal* is a database literal without variables. A *database fact* is a positive ground database literal. A *ground instance* of a database literal *L* is a ground database literal obtained by applying a ground substitution *i* to *L*. Analogously, a *constraint literal C* is an atomic formula with a constraint predicate symbol and ground instances of *C* are obtained by applying ground substitutions. A *literal* is either a database literal or a constraint literal.

A *safe conjunction* is a conjunction of constraint literals, positive database literals or possibly universally quantified negative database literals such that any variable occurring in the conjunction is either governed by a universal quantifier or occurs in a positive database literal. For example, the conjunction `place(P)∧∀K∀L(¬protection(P,K,L))` is safe, but not the conjunction `place(P)∧¬protection(P,K,L)`. The possibility of quantifying variables in negative database literals is useful when defining operations and goal-inference rules (see Sections 3.2 and 4.2); in the Prolog implementation, their evaluation is based on the closed world assumption [32]. The quantification of positive literals is more complicated to implement and is therefore not allowed.

In what follows, we use the notation $\forall\bar{x}(\neg L[\bar{x},\bar{y}])$ to indicate that $\neg L$ is a negative literal with variables partitioned into two lists, $\bar{x}$ and $\bar{y}$, such that the variables in $\bar{x}$ are universally quantified. Likewise, $\forall\bar{x}(\neg L[\bar{x},\bar{b}])$ denotes the result of substituting the variables in $\bar{y}$ by the constants in the list $\bar{b}$.

A *structure* M for L assigns to each symbol *s* of L a meaning s^M as usual. We assume that M assigns the usual meaning to constraint predicate symbols and constraint function symbols (such as the meaning assigned by arithmetic to numeric constraints). Given a formula σ and a set of formulae Σ of L, we use $\models_M \sigma$ to denote that M *satisfies* σ, or that σ *holds* in M, and $\Sigma \models \sigma$ to denote that σ is a *logical consequence* of Σ.

A *possible fact* of M is a database fact of L that holds in M. A *possible database state* of M is a set of possible facts of M.

We define the notion of a safe conjunction *holding* at a possible database state $\boldsymbol{s}$ as follows:

- a ground positive database literal *L holds* in $\boldsymbol{s}$ for *M,* denoted $\boldsymbol{s} \models_M L$, iff $L \in \boldsymbol{s}$
- a ground constraint literal *C* holds in s for M, denoted $\boldsymbol{s} \models_M C$, iff *C* is true in the usual interpretation
- a ground negative database literal $\neg L$ *holds* in $\boldsymbol{s}$ for *M,* denoted $\boldsymbol{s} \models_M \neg L$, iff $L \notin \boldsymbol{s}$
- a formula of the form $\forall\bar{x}(\neg L[\bar{x},\bar{b}])$ *holds* in $\boldsymbol{s}$ for *M,* denoted $\boldsymbol{s} \models_M \forall\bar{x}(\neg L[\bar{x},\bar{b}])$, iff, for all ground substitutions *i*, $\neg L[\bar{a},\bar{b}] \notin \boldsymbol{s}$, where $\bar{a}$ are the constants that *i* assigns to $\bar{x}$

- a safe conjunction of the form $L_1 \wedge ... \wedge L_n$ *holds* in $\boldsymbol{s}$ for M, denoted $\boldsymbol{s} \models_M L_1 \wedge ... \wedge L_n$, iff, for all ground substitutions i, for each $k \in [1,n]$, $\boldsymbol{s} \models_M L_k[i]$

The previous definitions outline the basic syntax and semantics of plot languages, but they do not capture the semantics of identifiers, attributes, relationships and the is-a relation. We therefore define a *plot static schema* as a first-order theory whose language is a plot language and whose axioms include the following:

- *ER model constraints:*
  - for each identifier defined for an entity class, an *identifier axiom* that captures the uniqueness property of the identifier
  - for each attribute defined for an entity or relationship class (including the Boolean and composite attributes), an *attribute axiom* that captures the domain and range of the attribute
  - for each relationship, a *relationship axiom* that captures which entity classes the relationship involves
  - *is-a axioms* indicating which classes are specializations of other classes
  - *role axioms* that constrain roles to denote sets of entity instances in the union of collections of entity sets (that play the role in the ER schema)
- *application domain constraints* that capture other properties of the application domain

The formal definition of axioms similar to the ER model constraints, framed in Description Logic, can be found in [5,6]. Furthermore, we note that the Prolog tool discussed throughout the paper was implemented to automatically take into account the ER model constraints.

Finally, a *model* of a plot static schema is a structure for the language of the schema that satisfies all axioms of the schema.

2.3 Prolog concrete notation for the static schema

In this section, we briefly explain the Prolog notation we adopt to express static schemas and database facts. The syntax and semantics follow that of standard Prolog. In particular, we use square brackets for conjunctive lists (with "," as separator) and regular parentheses for disjunctions (with ";" as separator).

An ER (static) schema is specified using Prolog clauses of the following patterns:

```
entity(<entity-class>,<identifier>).
relationship(<relationship-class>,[<entity-class>,...,entity-class>]).
attribute(<entity-class>,<attribute>).
attribute(<relationship-class>,<attribute>).
boolean(<attribute>).
composite(<attribute>,[<attribute-part>,...,<attribute-part>]).
is_a(<more-specialized-entity-class>,<more-general-entity-class>).
role(<role-name>,(<entity-class>;...;<entity-class>)).
```

A set of Prolog clauses of the above patterns has the double purpose of defining the alphabet of the plot language and indicating which ER axioms apply.

A database fact `L` is written as a Prolog clause of the form `db(L)`, where `db` is a special unary predicate symbol, and `L` is rewritten as a term with the help of above Prolog notation for lists. In more detail, we have:

Schema clause: `entity(<entity class>,<identifying attribute name>).`
Database fact: `db(<entity class>(<identifier>)).`

Schema clause: `attribute(<entity class>,<attribute name>).`
Database fact: `db(<attribute name>(<identifier>,<attribute value>)).`

Schema clause:
```
relationship(<relationship name>,
             [<entity class>,<entity class>]).
```
Database fact: `db(<relationship name>([<identifier>,<identifier>])).`

Schema clause: `attribute(<relationship name>,<attribute name>).`
Database fact:
```
db(<attribute name>([<identifier>,<identifier>],
                    <attribute value>)).
```

Schema clause: `role(<role name>,<entity class>).`
Database fact: `db(<role name>(<identifier>)).`

The static schema of our Swords and Dragons genre, in Prolog notation, becomes:

```
/* Static Schema */

1.   entity(creature,name).
2.   entity(person,name).
3.   entity(knight,name).
4.   entity(princess,name).
5.   entity(magician,name).
6.   entity(dragon,name).
7.   entity(place,place_name).
8.   is_a(person,creature).
9.   is_a(knight,person).
10.  is_a(princess,person).
11.  is_a(magician,person).
12.  is_a(dragon,creature).
13.  attribute(creature,nature).
14.  attribute(creature,strength).
15.  attribute(creature,alive).
16.  attribute(place,protection).
17.  boolean(alive).
18.  composite(protection,[kind,level]).
19.  relationship(home,[creature,place]).
20.  relationship(current_place,[creature,place]).
21.  relationship(acquaintance,[creature,creature]).
22.  relationship(married,[person,person]).
23.  relationship(kidnapped,[person,creature]).
24.  attribute(acquaintance,affection).
25.  role(hero,knight).
26.  role(victim,(princess;knight)).
27.  role(villain,(dragon;knight)).
28.  role(donor,magician).

/* Sample Database Facts */

29.  db(princess('Marian')).
30.  db(strength('Marian',10)).
31.  db(alive('Marian')).
32.  db(protection('White_Palace',[1,70])).
33.  db(acquaintance(['Marian','Hoel'])).
34.  db(affection(['Marian','Hoel'],0)).
35.  db(victim('Marian')).
```

Note that, according to the clause in line 14, `strength` is an attribute of `creature` and, in view of the clauses in lines 8 and 10, also an attribute of `person` and of `princess`. Hence, the database fact in line 30 is acceptable. Also note the special notation for the database facts in lines 31 and 32, in view of the declaration in line 17 of `alive` as a Boolean attribute and the declaration in line 18 of `protection` as a composite attribute. We stress that the Prolog tool interprets the clauses in lines 1 to 29 to automatically generate the ER model constraints for the static schema.

3. Dynamic schema

3.1 Informal description of the dynamic schema

The dynamic level of specification takes us from descriptions of states, which the static schemas cover, to *narratives*. While states are sets of facts, narratives are composed of events. An *event* is a transition between valid states, i.e., conforming to the established static integrity constraints. In addition, we require that the transition itself be valid, which means that it must obey a further set of restrictions, called *dynamic integrity constraints*.

We enforce both types of constraints by restricting state changes to what can be accomplished by applying a limited repertoire of pre-defined *domain-oriented operations*, as advocated in the abstract data types [23] and in the object-oriented [16] approaches. The operations must be *consistent* in such a way that, if they start on any valid initial state, their execution will always preserve all constraints. However, it is usually much harder to prove that the repertoire of operations is *complete*, that is, enough to allow that all intended valid states be reachable (from some initial state).

An analogous and entirely compatible notion has been proposed a long time ago in literary theory by the Russian researcher Vladimir Propp [31]. In order to specify the genre of fairy-tales, he described a set of 31 functions, comparable to what we are calling domain-oriented, or more appropriately here, genre-oriented operations, which he claimed to be enough to account for a large sample extracted from an anthology of fairy-tales compiled by Alekxandr Afanas'ev [2].

We equate the notion of *event* with the state change brought about by the execution of an operation. This understanding, which is in agreement with Propp's theory, has proved adequate for our purposes.

In order to formally specify operations, the *STRIPS – Stanford Research Institute Problem Solver* [17] method is very convenient, both for real-life domains and for fictional genres. Each operation is defined in terms of its pre- and post-conditions. *Pre-conditions* are conjunctions of positive or negative database facts, which must hold at the state in which the operation is to be executed. *Post-conditions*, or *effects*, consist of two sets of facts: those to be asserted and those to be retracted as a consequence of executing the operation. Such effects can be understood as the semantics of the operation. Furthermore, integrity preservation depends on a careful adjustment of the interplay among pre- and post-conditions over the entire repertoire of operations.

This interplay has an even more important consequence, which is to establish a partial order for the execution of the operations. Indeed, if the pre-conditions of an operation O_1 may only be satisfied by the post-conditions of another operation O_2, then O_2 should be executed before O_1. This leads, in turn, to a *backward chaining* strategy for plan generation. At this point, two comments are in order: (a) logic programming, as offered by Prolog, is appropriate for developing such planning algorithms; (b) constraint programming algorithms (as offered by the version of Prolog that we use [7]) provide a very powerful complement to logic inference for handling goals involving numerical attributes.

The notation for declaring the *signature* of operations should be extended in order to associate *pragmatic* information, especially concerning *agency*, with the parameters. For this purpose, Fillmore's *case grammar* proposal [18] is applicable. Out of the various choices of cases listed by different authors, we employ here*: agent, coagent, recipient, patient, object, and destination.* In the parameter list of an operation, each parameter is

characterized by a case, paired with either the entity class or role involved. Indicating a role, instead of an (entire) entity class, limits the participation in a case to those instances of one or more entity classes to which the role has been explicitly assigned. Notice that the case *agent* (and also *coagent*) introduces an *agent-oriented* [24] modeling view.

Section 5.3 elaborates on the dynamic schema for our Swords and Dragons genre.

3.2 Formalization of the dynamic schema

We first extend the notion of alphabet of a plot language to also contain the following sets of symbols, disjoint from the other symbols:

operation names	n-ary function symbols to denote operations
case names	constant symbols to denote cases

We then extend the syntax and semantics of plot languages to capture the properties of operations.

Let L be a plot language whose alphabet includes operation and case names. We first discuss the semantics of operations and then address case names. Given an operation name o of L, a structure M for L assigns to o a set o^M of pairs of database states.

An *operation specification* for an n-ary operation name o is an expression σ of the form $\{P\}o(t_1,\ldots,t_n)\{Q\}$, where

- $o(t_1,\ldots,t_n)$ is the *input declaration* of σ, where $t_1,\ldots,t_n$ is a list of terms
- P is the *pre-condition* of σ, defined as a safe conjunction
- Q is the *post-condition* of σ, defined as a conjunction of positive or negative database literals

The pre-condition defines when the operation can execute, whereas the post-condition indicates the facts (the positive literals) that must be asserted, and those that must be retracted (the negative literals) as a consequence of executing the operation.

Given a substitution i, we use $\sigma[i]$ to denote the *substituted version* of σ, obtained by uniformly applying i to the input declaration and to the pre- and post-conditions of σ. We say that σ is *ground* iff the input declaration, the pre- and the post-conditions of σ have no free variables.

Assume that σ is ground in this and the next definition. We say that a pair $(\boldsymbol{s},\boldsymbol{t})$ of possible database states in o^M *satisfies* σ for M, or that σ *holds* in $(\boldsymbol{s},\boldsymbol{t})$ for M, iff we have that

- $\boldsymbol{s} \models_M P$ (i.e., the pre-conditions are satisfied in $\boldsymbol{s}$ for M)
- $\boldsymbol{t} \models_M Q$ (i.e., the post-conditions are satisfied in $\boldsymbol{t}$ for M)
- for every possible database fact f of M, if neither f nor $\neg f$ occur in Q (which is ground by assumption), then $\boldsymbol{t} \models f$ iff $\boldsymbol{s} \models f$ (which is the frame requirement: preservation of satisfaction from $\boldsymbol{s}$ to $\boldsymbol{t}$ for ground database literals that are neither established nor negated by the post-condition Q)

We say that M *satisfies* σ, or that σ *holds* in M, iff σ *holds* in $(\boldsymbol{s},\boldsymbol{t})$ for M, for every pair $(\boldsymbol{s},\boldsymbol{t})$ of possible database states in o^M.

Assume now that σ is not necessarily ground. We say that M *satisfies* σ, or that σ *holds* in M, iff M satisfies $\sigma[i]$, for every ground substitution i of L (and likewise σ *holds* in pairs of database states). Note that, substitutions do not affect universally quantified variables that may appear in negative pre-conditions.

An operation is associated with an operation frame, which further constrains the semantics of the operation. A *case declaration* is an expression of the form $c:(e_1,...,e_m)$ where c is a case name and $(e_1,...,e_m)$ is a list of entity or role names. Given an n-ary operation name o, an *operation frame for* o is an expression of the form $o(c_1,...,c_n)$ where $c_1,...,c_n$ is a list of case declarations.

If an operation name o is associated with an operation frame of the form $o(c_1:(e_{11},...,e_{1m1}),...,c_n:(e_{n1},...,e_{nmn}))$ and an operation specification σ of the form $\{P\}o(t_1,...,t_n)\{Q\}$ then we redefine the notion that M *satisfies* σ, or that σ *holds* in M, iff M satisfies $\sigma[i]$, for every ground substitution i of L such that $t_k[i]^M \in e_{k1}{}^M \cup ... \cup e_{kmk}{}^M$.

An example of an operation frame and an operation specification from our running example would be the expressions f and $\{P\}o(t_1,t_2)\{Q\}$, where:

f = `reduce_protection(agent:(victim),destination:(place))`

$o(t_1,t_2)$ = `reduce_protection(V,P)  /* V is a victim and P is a place  */`

```
P = { current_place(V,P)∧     /* V is currently at P                    */
      protection(P,K,L)∧      /* P has protection of kind K and level L */
      nature(V,K)∧            /* the nature of V is K                   */
      L>0.0}                  /* the level of protection is positive    */

Q = {¬protection(P,K,L))∧     /* P no longer has protection K and L     */
      protection(P,K,L-10.0))}/* P now has protection K and L-10.0      */
```

Finally, we define a *plot dynamic schema* as a plot static schema augmented with operation and role names and with *dynamic axioms*, that is, operation specifications and operation frames. A *model* of a plot dynamic schema is again defined as a structure for the language of the schema that satisfies all axioms of the schema.

3.3 Prolog concrete notation for the dynamic schema

An operation is defined by two complementary Prolog clauses of the form:

```
operator_frame(<operator-id>,
      <operator-name>,
      [<case>: (<entity class or role>;...;<entity class or role>),...,
       <case>: (<entity class or role>;...;entity class or role>]).
operator(<operator-id>,
      <operator-name>(<parameter list>),
      <pre-conditions>,
      <post-conditions>,
      <estimated cost of operation>,
      <main effects>,
      [],[]).
```

The notion of main effects is new, conveys pragmatic information to the planner, and does not affect the semantics of the operation. The pre-conditions, post-conditions and main effects are expressed as conjunctive lists, using square brackets and "," as separator. The purpose of the last two parameters, shown above as empty lists, will be explained at the end of Section 4.3.

The sample operation specification introduced in Section 3.2, in Prolog notation, becomes:

```
operator_frame(2,reduce_protection,[agent:victim,object:place]).

operator(2,
     reduce_protection(V,P),/* V is a victim and P is a place          */
     [
       current_place(V,P),   /* V is currently at P                    */
       protection(P,[K,L]),  /* P has protection of kind K and level L */
       nature(V,K),          /* the nature of V is K                   */
       L>0.0                 /* the level of protection is positive    */
     ],
     [
       not(protection(P,[K,L])), /* P no longer has protection K and L */
       protection(P,[K,L-10.0])  /* P now has protection K and L-10.0  */
     ],
       10,                     /* (estimated cost of operation)      */
     [protection(P,[K,L-10.0])], /*(main effect of the operation)     */
     [],[]).
```

4. Behavioural schema

4.1 Informal description of the behavioural schema

The behavioural schema, as the name implies, defines the behaviour of animated agents, also represented as entity instances. It consists of a set of *goal-inference rules* that capture the *goals* that motivate the agents' actions when certain *situations* occur during a narrative. The behavioural schema may also contain a library of predefined plans, also called *complex operations*.

The plot composition algorithm uses the goal-inference rules as follows. At a given initial state, it first applies goal-inference rules to determine goals for the various agents. Then, it creates one or more plots to achieve the goals of each agent, whose combined effect may involve mutual interferences. The occurrence of events, caused by the simulated execution of the planned operations, will result in a new state wherein, again, the plot composition algorithm applies the goal-inference rules, until a state is reached where no new goal is inferred (or the user arbitrarily decides to end the process).

Willensky [39] has done a comprehensive study of positive and negative interferences between goals and plans, both of the same agent and of different agents. Negative interferences result in contradictions to be resolved, and positive interferences offer optimization opportunities. In both cases, different strategies can be employed to find how to alter goals and partial plots in order to obtain a consistent plot, in which even *failed* individual subplots may occur.

Finally, we remark that an alternative way to obtain plans to achieve the goals of an agent is to select, from an adequately structured library, a pre-existing typical plan, adapting it if necessary to specific circumstances. We consider the plan library to be part of the behavioural schema.

Taking typical plans as building blocks corresponds, in Artificial Intelligence terminology, to start from commonly used *scripts*, rather than constructing new plans from scratch with the primitive operations [34]. In literary terminology, there is a notion analogous to scripts, namely *types* and *motifs*, as in the monumental index compiled by Aarne and Thompson [1].

Section 5.4 elaborates on the goal-inference rules and typical plans for our Swords and Dragons genre.

4.2 Formalization of the behavioural schema

Let T be a plot dynamic theory with language L. Let M be a model of T. Recall that, to each operation name o of L, the model M assigns a set o^M of pairs of database states. Given a substitution i of L and an operation specification σ, also recall that $\sigma[i]$ denotes the *substituted version* of σ, obtained by uniformly applying i to the input declaration, the pre- and the post-conditions of σ. In particular, i may be a ground substitution of L.

A *temporal database* of a model M of T is a pair $T=(\boldsymbol{S},\boldsymbol{O})$ consisting of a sequence $\boldsymbol{S}=(\boldsymbol{s}_0,\boldsymbol{s}_1,\ldots)$ of possible database states of M and a possibly empty sequence $\boldsymbol{O}=(\boldsymbol{o}_1,\boldsymbol{o}_2,\ldots)$ of ground substituted versions of operation specifications of T such that ($\boldsymbol{S}$ starts on $\boldsymbol{s}_0$ and $\boldsymbol{O}$ starts on $\boldsymbol{o}_1$, by convention; if $\boldsymbol{O}$ is empty, then T reduces to just one database state):

- $|\boldsymbol{S}| = |\boldsymbol{O}| + 1$ (the length of $\boldsymbol{S}$ is the length of $\boldsymbol{O}$ plus one)
- for each $i\in[1,|\boldsymbol{O}|]$, $\boldsymbol{o}_i$ holds in $(\boldsymbol{s}_{i-1},\boldsymbol{s}_i)$ for M (non-initial states are caused by ground substituted versions of operation specifications of T)

Given a temporal database $T=(\boldsymbol{S},\boldsymbol{O})$ of M, and $k\in[0,|\boldsymbol{S}|]$, the k^{th} *continuation* of T is the temporal database $T^k=(\boldsymbol{S}^k,\boldsymbol{O}^k)$ such that $\boldsymbol{S}^k$ and $\boldsymbol{O}^k$ are the suffixes of $\boldsymbol{S}$ and $\boldsymbol{O}$ starting on the k^{th} element of $\boldsymbol{S}$ and $\boldsymbol{O}$, respectively. Note that a continuation $T^k=(\boldsymbol{S}^k,\boldsymbol{O}^k)$ of T is indeed a temporal database, and that the temporal database consisting of just the last state of T and the empty sequence of operations is the n^{th} continuation of T, where $n=|\boldsymbol{S}|$.

Temporal formulae are expressions recursively defined as for first-order languages, with one additional syntactic rule:

- if α and β are temporal formula, then so are the expressions:

$\Diamond\alpha$	α eventually holds
$\Box\alpha$	α always holds
$\bigcirc\alpha$	α holds next
$\alpha \cup \beta$	α holds until β

We define the notion of *holding* at a temporal database $T=(\boldsymbol{S},\boldsymbol{O})$ of M, as follows:

- a first-order formula α *holds* in T for M, denoted $T \models_M \alpha$, iff $\boldsymbol{s}_0 \models_M \alpha$ (that is, α holds in the first state $\boldsymbol{s}_0$ of T)
- a formula of the form $\Diamond\alpha$ *holds* in T for M, denoted $T \models_M \Diamond\alpha$, iff for some $k\in[0,|\boldsymbol{S}|]$, $T^k \models_M \alpha$ (that is, α holds in some continuation T^k of T)

- a formula of the form $\Box\alpha$ *holds* in T for M, denoted $T \models_M \Box\alpha$, iff for all $k \in [0,|\boldsymbol{S}|]$, $T^k \models_M \alpha$ (that is, α holds in all continuations T^k of T)
- a formula of the form $\bigcirc\alpha$ *holds* in T for M, denoted $T \models_M \bigcirc\alpha$, iff $T^1 \models_M \alpha$ (that is, α holds in T^1, the continuation starting on the second state of T)
- a formula of the form $\alpha \mathbf{U} \beta$ *holds* in T for M, denoted $T \models_M (\alpha \mathbf{U} \beta)$, iff there is $p \in [0,|\boldsymbol{S}|]$ such that $T^p \models_M \beta$ and, for all $q \in [0,p)$, $T^q \models_M \alpha$ (that is, α holds in all continuations until reaching, but excluding, a continuation where β holds)

We extend the notion of holding at a temporal database to complex temporal formulae as usual.

A *goal-inference rule* is recursively defined as follows:

- a temporal formula of the form $\Diamond \boldsymbol{L}$ is a goal-inference rule, where $\boldsymbol{L}$ is a safe conjunction
- a temporal formula of the form $(\boldsymbol{L} \Rightarrow \Gamma)$ or of the form $\Box(\boldsymbol{L} \Rightarrow \Gamma)$ is a goal-inference rule, where $\boldsymbol{L}$ is a safe conjunction and Γ is a goal-inference rule

The quantification of variables in goal-inference rules obeys the following rules:

- the variables occurring only within the scope of a temporal operator $\Diamond$ are locally existentially quantified; and
- all other variables are globally universally quantified.

Using these rules, quantifiers can be left implicit in the goal-inference rules. For example, the formula $\Box(\boldsymbol{C_1} \Rightarrow \Box(\boldsymbol{C_2} \Rightarrow \Box(\boldsymbol{C_3} \Rightarrow \Diamond \boldsymbol{G})))$ is a goal-inference rule. Due to the implications used, the formula holds at a temporal database $T=(\boldsymbol{S},\boldsymbol{O})$ of M iff, whenever there are three states $\boldsymbol{S}_{c1}$, $\boldsymbol{S}_{c2}$, $\boldsymbol{S}_{c3}$, with $c_1 \le c_2 \le c_3$, such that $\boldsymbol{C_i}$ holds in $\boldsymbol{S}_{ci}$, for i=1,2,3, then there must be a state $\boldsymbol{S}_g$ with $c_3 \le g$, such that $\boldsymbol{G}$ holds in $\boldsymbol{S}_g$. Intuitively, $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, $\boldsymbol{C_3}$ define a sequence of conditions that, if satisfied, implies that the goal $\boldsymbol{G}$ must be satisfied in a future state (with respect to the state where $\boldsymbol{C_3}$ is satisfied).

Examples of goal-inference rules would be:

```
/* A hero wants to become stronger than the villain */

      (villain(VIL)∧ strength(VIL,Lv) ∧
       hero(HERO) ∧ strength(HERO,Lh) ∧
       ⇒ ◊(strength(HERO,LS) ∧ LS > Lv)

/* If victim is kidnapped, hero will want to rescue her */

     □(kidnapped(VIC,VIL)⇒ ◊¬kidnapped(VIC,VIL))
```

A *plot* is a triple $P=(\boldsymbol{L},O,p)$ where

- $\boldsymbol{L}$ is a safe conjunction
- O is a set of operation specifications
- p is a partial order on O

A plot P is *ground* iff all operation specifications of P are ground. The *meaning* of a ground plot $P=(\boldsymbol{L},O,p)$ in M is the set P^M consisting of all temporal databases $T=(\boldsymbol{S},\boldsymbol{O})$ of M such that

- $\boldsymbol{s}_0 \models_M \boldsymbol{L}$ (the initial state $\boldsymbol{s}_0$ of $\boldsymbol{S}$ satisfies the constraints of P)
- $\boldsymbol{O}$ is a sequence consisting exactly of the operation specifications in O in an order consistent with the partial order p

The *meaning* of a non-ground plot $P=(\boldsymbol{L},O,p)$ in M is the set P^M is the set of all temporal databases T such that i is a substitution of L and $T \in P[i]^M$.

We note that Ciarlini et al. [11] describe a computational method for checking the satisfaction of goal-inference rules with respect to a plot without constructing the corresponding temporal databases. We also refer the reader to [10,11] for the formalization of complex operations, which is outside the scope of this paper.

Finally, we define a *plot behavioural schema* as a plot dynamic schema augmented with goal-inference rules and a set of complex operations. A *model* of a plot behavioural schema is again defined as a structure for the language of the schema that satisfies all axioms of the schema.

4.3 Prolog concrete notation for the behavioural schema

The Prolog concrete notation for condition-goal rules uses timestamp variables to capture the semantics of the temporal operators introduced in Section 4.2. The translation from our formal notation to the Prolog concrete notation is however straightforward.

As an example of a goal-inference rule in Prolog notation, we have:

```
/* If victim is kidnapped, hero will want to rescue her */
rule( [e(T1,kidnapped(VIC,VIL))],
       ([T2],[ h(T2,not(kidnapped(VIC,VIL))),h(T2>T1)],true) ).
```

Note that this Prolog clause corresponds to the following rule, introduced in Section 4.2:

$\Box$(kidnapped(VIC,VIL) $\Rightarrow$ $\Diamond\neg$kidnapped(VIC,VIL))

Complex operations have the same syntax as basic operations, introduced in Section 3.3. If a complex operation results from a *composition* of other possibly complex or basic operations, the two last parameters (shown as empty lists in the `operator` clause pattern of section 3.3) will contain, respectively, the list of component operations, each with a different f_i tag, and a list of tag pairs (f_i , f_j) declaring any order requirements holding between the operations.

5. Example: a Swords and Dragons genre

5.1 Example Scenario

The example scenario consists of an ample field, on which certain landmarks can be distinguished. These are the White Palace, the Gray Castle, the Red Castle, the Church and the Green Forest. The White Palace is the home of Princess Marian and also houses a temporary visitor, a knight called Hoel. The Gray Castle is the home of Hoel and of Brian, an even worthier knight. The Red Castle is occupied by Draco, a flying dragon. In the Green Forest lives the magician Turjan. The White Palace and the Red Castle are protected by armed guardians, and the Green Forest by magical trees; the other places, including the Gray Castle, have no such defenses.

These creatures, both the persons and the dragon, can be described in terms of their good or evil nature and of their strength. As to nature, the princess and the knights are reputed to be on the side of goodness, whereas the dragon is evil; contrasting with all others, the magician is neutral. In the beginning, unsurprisingly, all creatures are alive, and no one is stronger than the dragon. Differently from these leading characters, the protecting guardians figure as mere extras, individually undistinguishable. Relevant only in groups, they are a feature of the places they are charged to protect, and the protection afforded is characterized by the size of the group and by kind (which reflects the nature of the place-owners).

The inter-personal relations are simple. All creatures are acquainted with each other, but demonstrate no mutual feelings initially, except for the two knights, who have a strong positive affection for the princess. At a later time, one of the heroes and the princess may eventually get married. On the negative side, the dragon may subsequently kidnap the princess, and keep her under custody. The creatures are all in their homes at the beginning (with the single exception of Hoel), and the princess, the knights and the dragon are normally free to be at different places in other occasions; the magician, however, is confined to his sylvan refuge.

In our limited Swords and Dragons genre, actions are mostly physical. Heroes, villains and even victims are able to fight and take measures to raise their chance of victory. Before engaging in personal battle, the attacker often has to penetrate through the group of guardians surrounding the prospective victim’s present location; this may be quite hard, unless the victim foolishly dismisses a number of guardians. And the combat proper will consume the energies of a fighter. Is the attacker's strength enough to defeat and kill the adversary? If not, he should seek a powerful magician to obtain a surplus of fighting power.

But, as donors tend to be in folktales, a magician is a capricious being, easily ill-disposed when approached without due courtesy. He may then pretend to yield to the hero’s request but will in fact reduce his strength to the bare minimum necessary to start a combat – just to be inevitably defeated in the sequel.

Heroic knights are destined to love damsels, who in turn may not respond to their entreaties at the beginning. But, if a villain kidnaps a princess and a hero successfully frees her, then gratitude and admiration should change her inclination.

Many actions are closely associated with places. So, to say that a villain kidnaps a victim means that he brings her to his lair, and marriage can only be celebrated at a church. All characters, except donors, continually move across the scene to accomplish their missions.

The various characters are motivated to act by their inner drives. Typically, a knight like Brian is anxious to be invested with superior heroic force, so that some day he can become a dragon-slayer. By contrast, Princess Marian does not even imagine that there may be any possibility of violence, and she finds no use for the presence of so many guards around her palace.

Draco is continually in the alert for signs of a weakening in her protection, awaiting a chance to come and achieve the maiden's abduction. Attempts to kidnap may meet resistance, with considerable risk to the victim. On purpose or by accident, the dragon may end up killing his fragile prey.

Depending on the outcome of the villainy − abduction or death of the princess − one hero, or both, are impelled to either rescue or, in the worst case, avenge her. If released alive from captivity, the princess will be full of tender feelings for her saviour. Both would love each other and would thus be anxious to have their marriage celebrated.

If the two knights participate of a heroic quest on behalf of the princess, they may or may not collaborate. They both love her, and are bound to compete, loyally or not, to win her hand.

Finally, the magician Turjan does not seem to wish anything. He stands still in the forest, where people sometimes seek him. The heroes come to demand a gift of fighting energy and his reaction depends on how he is disposed toward the newcomer. Desiring nothing, he never initiates any plans. But, when one least expects, he can with a gesture transmute a kind person into a powerful evil creature.

5.2 Description of the mini-world – static schema

Figure 1 displays the static schema. Briefly, we have:

- `creature`, an entity class, identified by `name`, with attributes `nature`, `strength`, `gender` and `alive` (of Boolean type)
- `person` and `dragon`, specializations of `creature`
- `princess`, `knight`, and `magician`, specializations of `person`
- `place`, an entity class, identified by `place_name` with a composite attribute `protection`, composed of `kind` and `level`
- `home` and `current_place`, two n-1 relationships between `creatures` and `places`
- `acquaintance`, a relationship involving `creature` twice, with attribute `affection`
- `married`, a 1-1 relationship involving `person` twice
- `kidnapped`, a n-1 relationship between `persons` and `creatures` (more than one `person` can be simultaneously held by one kidnapper)
- `hero`, `victim`, `villain` and `donor`, the roles adopted

For our purposes, we did not specialize `place`, but this is largely a matter of taste; one might readily come up with a variety of distinguishing criteria applicable to our scenario.

The choice of a convenient type for attribute values is crucial. For example, one could at first consider `good` and `evil` as possible values for `nature`, as well as for `kind` of `protection`. We preferred instead `1` and `-1`, which permits their use in various arithmetic comparison formulas, involving `strength` and `level` of `protection` (as will be seen in Section 5.3). An even more important choice was for the type of `affection`. Again, the

intuitive preference might be some word indicating, for a pair of creatures *A* and *B*, in this order, the current feeling of *A* for *B*. Here, our choice was motivated by what is practically a consensus in affective computing [38] research: drives and emotions are better expressed as points in numerical scales within a given range (typically from 0 to 100). This makes it easier to describe gradual increases and decreases in emotional intensity. Also, we decided to allow zero and negative values to denote, respectively, neutral and adverse feelings. Finally, in order to take advantage of the real-number constraint programming package of the Prolog version used, we write all numbers as reals, although we are only concerned with integer values.

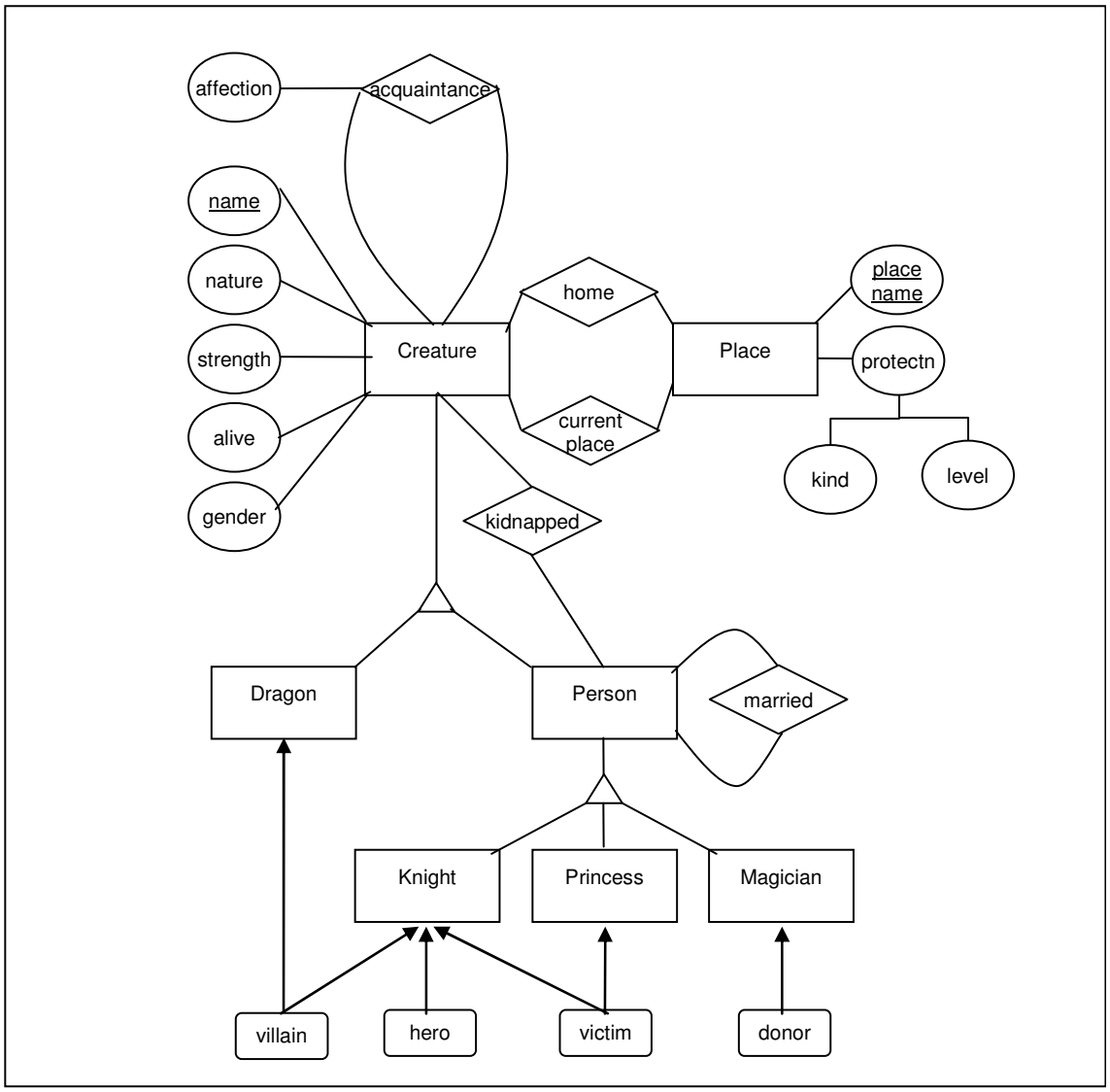

Figure 1: Entity-Relationship diagram

Our choice of roles – hero, victim, villain and donor – is a subset of the seven *dramatis personae* proposed by Vladimir Propp [31] for Russian fairy-tales. Roles hero and donor are here assigned only to knights and magicians, respectively. On the other hand, although it is more natural to assign the role of victim to a princess, and that of villain to a dragon, we also allow in our specified genre that knights may figure as victims or villains. Any entity instance to which one or more roles have been assigned is a character (one of the *dramatis personae*) in the story.

A number of static integrity constraints are assumed. The most obvious is that whatever attribute a creature may have should only retain any significance while it is alive. All attributes here are single-valued. If a creature is playing the role of villain, his nature must be -1, whereas heroes and victims, who act as "good" characters, are rated 1. Thus, in view of the single-valuedness of attributes, a knight can be at the same time hero and victim, but not hero and villain. A donor does not take sides, his neutrality being marked by an intermediate 0 value. Reflecting the inclination of the owners, the kind of protection of the several places coincides with the nature of the characters who make them their home.

As the diagram in Figure 1 shows, but not our Prolog clauses, only relationship acquaintance is unrestricted; the others are either 1-1 or 1-n, which constitutes an obvious static constraint. For a magician, here playing the role of donor, the current_place must coincide with his home at every state, a restriction that does not affect the other creatures. Moreover, married can only hold between persons of opposite gender.

A genre is of course compatible with an ample choice of (valid) initial states. Different initial states lead to the creation of possibly very different narratives, all of which are constrained to remain within the limits of the defined genre. In our sample scenario, we assume that, in the initial database, the villainous Draco is stronger than the two knights, of whom Brian is the most vigorous, and that the potential victim, Princess Marian, is indifferent to both knights, despite their perfect love (100 in affection) for her. True to his role as donor, Turjan the Archmage is neither good nor evil (0 for nature).

The closed world assumption [32], familiar to database practioners, justifies the conclusion that no one is married or kidnapped at the initial state, simply because no such facts are explicitly recorded in the database.

Appendices I and II contain, respectively, the full Prolog code for the static schema and for one of the initial states used in our experiments.

5.3 Events that change the mini-world – dynamic schema

Our Swords and Dragons genre has ten event-producing operations:

```
1. go(CH,PL)
2. reduce_protection(CH,PL)
3. kidnap(CH1,CH2)
4. attack(CH,PL)
5. fight(CH1,CH2)
6. kill(CH1,CH2)
7. free(CH1,CH2)
8. marry(CH1,CH2)
9. donate(CH1,CH2)
10. bewitch(CH1,CH2)
```

All operations share one evident pre-condition: the agent must be alive. Most operations also require that the agent must not be in a kidnapped status, wherein his freedom to act would be necessarily limited. For operations involving two characters, both must be in the same current_place. Operations involving a physical confrontation are only admitted between characters of opposite nature. A mandatory post-condition is that, when an attribute is modified to receive a new value, the list of effects always prescribes the exclusion of the old value, since all attributes are single-valued in our example. Specific characteristics for each operation are reviewed below:

- The agent of operation go(CH,PL) can be any character, except a donor; the destination is of course a place. A pre-condition is that the character CH should neither be currently kidnapped (a general requirement, as said above) nor be keeping someone kidnapped. Presumably the kidnapper must be constantly vigilant, to counter any attempt towards the victim's liberation. The effect of the operation is to make PL the current_place where CH is.
- Only the potential victim can imprudently dismiss some of the guardians of the place where she currently is, as agent of the reduce_protection(CH,PL) operation, whose object is a place. The current number of guardians serving as sentinels must be positive, and each execution of the operation reduces it by a factor of 10 (written as 10.0, in the required real-number format). The exact decrement will be determined at the *dramatization* stage.
- Villain and victim are the roles assigned to CH1 and CH2, the agent and the patient, respectively, of operation kidnap(CH1,CH2). A vital pre-condition is that the strength of the villain be enough to break into the place where the victim is. The formula for the comparison says that his strength should be greater than that of his victim, added to the level of protection of the place. But the kind of protection is also taken into consideration, being multiplied by the level (remember that kind is a number, 1 or -1, to indicate whether the guardians are either on the side of goodness or of evil). As a result, if the victim is currently in a place dominated by evil, the level of protection will actually be subtracted from her strength. Kidnapping results in the victim being imprisoned in the home of the kidnapper.
- A hero, not currently kidnapped (recall that the same individual who plays the role of hero can simultaneously be a victim), or a villain can be the agent of attack(CH,PL) intent on decimating the group of guardians protecting PL, which stands as the object of the action. The nature of the agent must be the contrary of the kind of protection of the attacked place. The current level of protection (associated with the number of guardians), which must be positive, is reduced by a factor of 30. The operation has the side-effect of displeasing those who have their home in PL: their affection for the attacker now becomes strongly negative (-100).
- Two characters of opposite nature, but never a donor, currently having strength of at least 10, can play the agent and coagent of fight(CH1,CH2). The level of protection of the place where the combat happens must be null or negative; so the troop protecting such locations must first suffer an attack, before the leading

characters can face each other. The confrontation is extenuating for both participants, which is indicated by the mutual subtraction of their strengths as a result.

- Agent and coagent of kill(CH1,CH2) are as in the preceding operation. The killer's strength must be strictly greater than 10; and the character killed must either no longer be able to fight or have the bare minimum necessary for that, which is expressed by requiring that his strength be at most equal to 10. The obvious effect is that CH2 is no longer alive.
- Operation free(CH1,CH2) can be performed by a hero, to the benefit of a kidnapped victim, only after the kidnapper is dead. Besides the effect that CH2 is no longer kidnapped, the operation has the virtue to raise to the maximum value (100) the affection of the grateful victim for her liberator.
- In our version of marry(CH1,CH2), the agent CH1 must be a hero and the coagent, CH2, a victim, usually representing the proverbial motif of the "maiden in distress rescued by a loving knight". Their mutual affection has to be greater than 80 (note this might already be true at the initial state, but then there would be no need for heroic action). They must be originally single. To acquire the married status, their presence at the Church is required.
- The first operation whose agent must exclusively be a donor, a role that is reserved to magicians in our genre, is donate(CH1,CH2), whereby the recipient, always a hero, is given an amount of fighting power. The measure of the new strength of CH2 depends on how he approaches the donor CH1. A courteous attitude is rewarded with an increase of 80 above his current strength, whereas rudeness, demonstrated by a previous attack against the defenses of the magician's home, is punished by having his strength set to the minimum required for fighting (10), regardless of what the previous value was.
- The second operation having a donor as agent, namely bewitch(CH1,CH2), has as patient either hero or victim, which are the two classes of characters normally endowed with a good nature. The dreadful double effect of the operation is to instill an evil nature into CH2 who, at the same time, is made very strong (a strength of 100).

It is worksome but not too hard to check how the combined interplay of pre-conditions and post-conditions in this repertoire contributes to the preservation of the static and dynamic integrity constraints, once the validity of the postulated initial state has been verified. As a trivial example with a static constraint, one can readily see that, at every state reachable from the initial state through the operations, the current place of the donor is invariably his home, provided that this was true at the initial state.

Killing an enemy is a task requiring wise tactics, in view of the dynamic constraints involved. If CH1 intends to kill CH2, he may or may not have to fight beforehand, so as to reduce the strength of the adversary. Value 10 is especially critical in this regard: it is not sufficient for CH1 as prospective killer, whereas CH2 can be killed if he has this value (or less). So, there is no need to fight if CH2 already has strength no greater than 10. On the other hand, 10 is the minimum required to start fighting, which may induce an ill-advised

character to challenge another with no chance to win (recall how the discourteous `recipient` is treated in the `donate` operation described above).

Now let us examine what happens when fighting takes place. Clearly only the situation wherein `CH1` is stronger than `CH2` needs to be considered. Suppose `CH1` has `strength` 30 and `CH2` has 20. As indicated as an effect of the energy-consuming `fight` operation, the `strengths` of the two opponents are subtracted from each other, so `CH1` ends up with 10 and `CH2` with -10. As a consequence, `CH2` can now be killed – but not by `CH1`, who became too weak for that. (Notice that the same happens with `strengths` of 20 and 10 respectively, which is ironical, since in this case `CH1` could have dispatched the enemy directly without fighting).

As an even subtler dynamic constraint, observe that, once `kidnapped`, a `victim` has no way to escape from custody by her own action, inevitably needing the initiative of one or more `heroes`. When dealing with fiction, one is allowed to make certain assumptions that may seem unrealistic. One of the general principles governing the genesis of fictional stories is that *functional events* [3,14] should be included, plausible or not entirely so, as a prompt to adventurous deployments. For instance, this "maiden in distress" situation works as an inducement for heroic quest.

In our example specification, if one starts from a valid initial state and only the nine first operations above are used, the generated plots should conform to all constraints and be recognizable as legitimate representatives of the intended genre. The pre- and post-conditions of these operations were carefully balanced for that. However, if the tenth operation – `bewitch` – happens to be utilized, this may no longer be true. The introduction of a disturbing element serves a purpose here: to create the possibility of `transgressing` some of the conventions of the genre, such as the understanding that all participants retain their `nature` throughout their lives. Again, fiction has a latitude that one would hardly admit in business application domains.

Appendix III contains the full Prolog code for the dynamis schema.

5.4 Motivation of the dramatis personae – behavioural schema

In this section, we first introduce the goal-inference rules and then address the repertoires of pre-existing plans for our Swords and Dragons genre. The definition of each rule, in the notation of Section 4.2, is followed by a brief discussion.

- The first two in our example are activated right at the initial state. The first rule refers to the `heroes`. At least one `hero` should be prepared for future missions and so, if there exists some `villain` stronger than him, he will try to acquire an even superior `strength`.

```
/* (1) The strongest hero wants to become stronger than the villain
*/
 (villain(VIL)∧ strength(VIL,Lv) ∧
  hero(HERO) ∧ strength(HERO,Lh)
  ⇒ ◊(strength(HERO,LS) ∧ LS > Lv)
```

- The second rule applies to the victim. It is very common in folktales that a victim can be blamed as partly responsible for the villainy that she will suffer. As Propp observed, her *complicity* is revealed as she, for example, exposes herself by weakening the defenses surrounding her. Accordingly, the rule assesses the initial level of protection of the place where she is, and sets its reduction as a goal. As already seen in the pre- and post-conditions of the operations, the nature of the victim and the type of protection of the place appear as coefficients, affecting the sign of the terms in the inequality. A different variable, PLACE2, denotes the location of the victim at future time; this allows *two* possibilities for achieving less protection: the planner can either apply (one or more times) the reduce_protection operation to the original PLACE1 – in which case the two variables will be treated as identical – or can cause the imprudent maiden to go to some different location already offering an inferior protection.

```
/* (2) Victim reduces protection of her current location
*/
 (victim(VIC) ∧ nature(VIC,KIND0) ∧
  current_place(VIC,PLACE1) ∧
  protection(PLACE1,KIND1,PROT1))
  ⇒ ◊(current_place(VIC,PLACE2) ∧
       protection(PLACE2,KIND2,PROT2) ∧
       KIND2*KIND0*PROT2 < KIND1*KIND0*PROT1)
```

- If the goal of the second rule is reached, the third rule is triggered, producing in the villain a desire to take advantage of the weaker condition of the victim, by having her kidnapped. Although this is the type of villainy that determines the normal continuation of the plot, it may happen instead that the villain perpetrates a different villainy, by murdering the victim. To cover this circumstance, it became necessary to add to the situation part of the rule the seemingly redundant requirement that the victim needs to be still alive if the villain proposes to have her kidnapped. Without this additional requirement, we would have a goal conflict with the fifth rule (described later).

```
/* (3) If victim's protection is reduced, villain will want to
   kidnap her */

 (victim(VIC) ∧ nature(VIC,KIND0) ∧
  current_place(VIC,PLACE1) ∧
  protection(PLACE1,KIND1,PROT1))
  ⇒ □(current_place(VIC,PLACE2) ∧
       protection(PLACE2,KIND2,PROT2) ∧
       KIND2*KIND0*PROT2 < KIND1*KIND0*PROT1
        ⇒ ◊(alive(VIC) ∧
             kidnapped(VIC,VIL)))
```

- The fourth rule says that, if kidnapping has occurred, the goal of reverting this situation will arise. The rule does not explicitly refer to the heroes as the necessary

agents who can accomplish the deed, but the overall specification of the genre calls for their participation, effectively excluding any other character.

```
/* (4) If victim is kidnapped, hero will want to rescue her */
□ (kidnapped(VIC,VIL) ⇒ ◊¬kidnapped(VIC,VIL))
```

- The fifth rule applies to a situation in which the villain has performed the action of killing the victim. All that remains for the heroes (once more not explicitly mentioned) to do is to vindicate her death, by making the villain lose his life. If both this rule and third rule were activated at the same occasion, a contradiction would result: the goal that the villain be not_alive makes it impossible to execute operation kidnap, required to satisfy the goal of rule three. Evidently the motivating situations for the two rules are mutually exclusive and so they should never be simultaneously active, since it does not make sense to kidnap a dead victim – but we find useful to report this as a problem, to illustrate how crucial a careful analysis of the specification is. Indeed, at an early design phase, we overlooked the necessity to spell out in the situation part of rule three that the victim should be alive, and took some time to realize what was causing trouble to the plan generator.

```
/* (5) If victim is killed, hero will want to avenge her */
□((victim(VIC) ∧ villain(VIL) ∧ kill(VIL,VIC))⇒ ◊¬alive(VIL))
```

- The sixth and last rule, if ever activated, will lead the plot to a happy ending: if two persons love each other with perfect love (or almost perfect, since the required affection is merely 95), and are still single, they will want to get married. That the married attribute for each person is tested in one direction only should not sound peculiar: operation marry asserts the attribute in both directions (and, as always, we must rely on the correctness of the initial state for complete information about already married people). Note also that the combined effect of the specification clauses restricts marriage to a hero and a victim, roles that are respectively assigned in the example initial state to each knight and to the princess, thus enforcing the opposite gender requirement.

```
/* (6) If the affection between two persons is high, then they will
  want to get married */
□ ((affection(CH1,CH2,L1) ∧ affection(CH2,CH1,L2) ∧
    ¬married(CH1,X) ∧ ¬married(CH2,Y) ∧ L2>95.0 ∧ L1>95.0)
   ⇒ ◊married(CH1,CH2))
```

Appendix IV contains the full Prolog code for the goal-inference rules included in the behavioural schema.

At the end of section 4.1, we mentioned, as an alternative or complementary strategy for goal achievement, the possibility of resorting to repertoires of pre-existing plans, whose utilization further characterizes the observed behaviour of the various agents. Such typical plans (or complex operations) are organized in *is-a* and *part-of* hierarchies. Figure 2 shows both hierarchies for our running example, where single arrows denote composition (*part-of* link) and double arrows denote generalization (*is-a* link) (we refer the reader to [25] for a

detailed description of the notation adopted). A sketchy top-down description follows, with levels numbered from 0 to 4 (the bottom level is occupied by the basic operations introduced in Section 5.3).

- Level 0 – `adventure`: Located at the root position, operation `adventure` has components: `do_villainy`, `retaliate`, `accompany` and `donate`. It specializes into: `rescue` or `avenge`.

- Level 1 – `rescue`, `avenge`: These are the two specializations of `adventure`. The `rescue` variety has components: `abduct`, `liberate`, `marry`, `accompany`, `donate`. The other variety, `avenge`, has components: `murder`, `execute`, `accompany`, `donate`. As Figure 2 shows, there are connecting edges to only some of the components; such edges are unnecessary for `accompany` and `donate`, which are *inherited* from `adventure` via the *is-a* link. Note that, for both `rescue` and `avenge`, the *is-a* inheritance mechanism would also indicate `do_villainy` and `retaliate` as components – but the existence of direct edges to specific forms of `villainy` and `retaliation` (the pair `abduct`, `liberate` for `rescue`, the pair `murder` and `execute` for `avenge`) in fact overrules the *is-a* implicit inheritance discipline. In other words, one can say that the choice of a villainy *preempts* the choice of the appropriate retaliation.

- Level 2 – `do_villainy`, `retaliate`, `accompany`: Operation `do_villainy` specializes into: `abduct` or `murder`; `retaliate` specializes into: `liberate` or `execute`; `accompany` specializes into: `help` or `false_help`. Names are, as usual, a matter of personal preference, but we tried our best to select meaningful words; `accompany`, for example, evokes the convention, pointed out by folklorists, that certain persons who aid (or hinder) the `hero` in his mission march by his side (playing the role of `helpers` or of `false_heroes`), while others (the typical `donors`) usually stay behind and take no part in the main action.

- Level 3 – `abduct`, `murder`, `execute`, `liberate`, `help`, `false_help`: Both `villainies` have a first component that signals the complicity of the `victim`. So, `abduct` has components: `reduce_protection`, `attack`, `kidnap`; while `murder` has components: `reduce protection`, `attack`, `fight`, `kill`. Both retaliations involve killing the `villain`, and include all preparatory actions which may or may not be needed in view of current circumstances. Variety `liberate` has components: `attack`, `fight`, `kill`, `free`, whereas `execute` has components: `attack`, `fight`, `kill`. Sincere helpers can contribute in various ways, not necessarily doing all actions listed here, and noting that `kill` should rather be reserved as a prerogative of the main `hero`. A clever `false_helper` is likely to enter the battlefield only when the struggle is over, and surreptitiously open the doors of the dungeon to the `victim`, thereby seducing her with an eye to matrimony. Thus, `help` has components: `attack`, `fight`, `free`. Effortless `false help` has components: `free`, `marry`.

We left out two basic operations (level 4) from this hierarchy. Pervasive as it is when physical events are contemplated, operation `go` is in fact an ultimate component of practically all actions, and therefore is assumed to be present even if not indicated explicitly. On the other hand, `bewitch` was deliberately excluded. As noted before, plots including `bewitch` are to be considered *transgressive* rather than typical in the context of our genre. They reveal the `magician's` inclination to subvert an until then innocent world, by acting as a *trickster*.

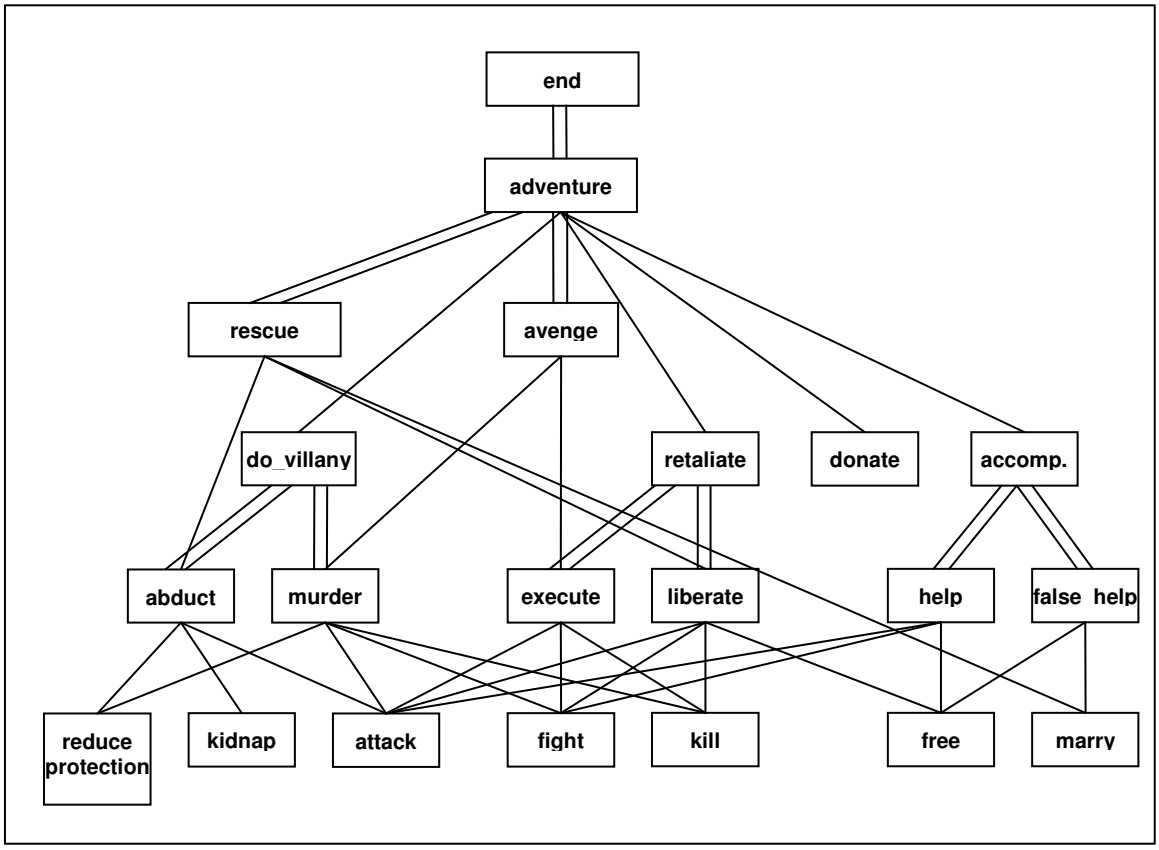

Figure 2: hierarchy of typical plans

5.5 Example of a plot generated by LOGTELL

We developed a prototype tool, **LOGTELL**, for interactively generating and dramatizing stories, through alternating stages of goal inference, planning, user intervention and 3D visualization [13]. **LOGTELL** incorporates a plan generator, adapted from **Abtweak** [40], and provides two main mechanisms to handle goal abandonment and competitive plan execution: *conditional goals* and *limited goals* [11]. A conditional goal has attached to it a *survival condition*, which the planner must check to determine whether the goal should still be pursued. Limited goals are those that have an associated *limit* (expressed as a natural number). The limit restricts the number of events that can be inserted to achieve the goal. Other strategies are being considered for future inclusion in our method.

To handle pre-existing typical plans in **LOGTELL**, we introduced a library structure that allows *plan-recognition* by a method proposed by Kautz [26]. The method consists of

matching a few intended events against the library, trying to find one or more plans of which the intended events may be part. This additional feature also serves to guide the user in his manual interventions, since, once a retrieved plan is found and displayed, the user can determine the insertion of one or more of its events into the plot being composed [13].

An example plot generated by the **LOGTELL** prototype is shown in Figure 3. The contents of the boxes indicate the executed operations (and also the goals, in "gen_goal" clauses) prefixed by numerical tags, used internally to record the partial order requirements. The connecting edges are manually inserted by the user to choose a fully linearized sequence compatible with the partial order requirements, which must be done as a preliminary step to *dramatization*.

Upon traversing the plot, a simple-minded template-based facility can "read" it and produce the coarse text of Figure 4. The resulting animation is illustrated in Figure 5.

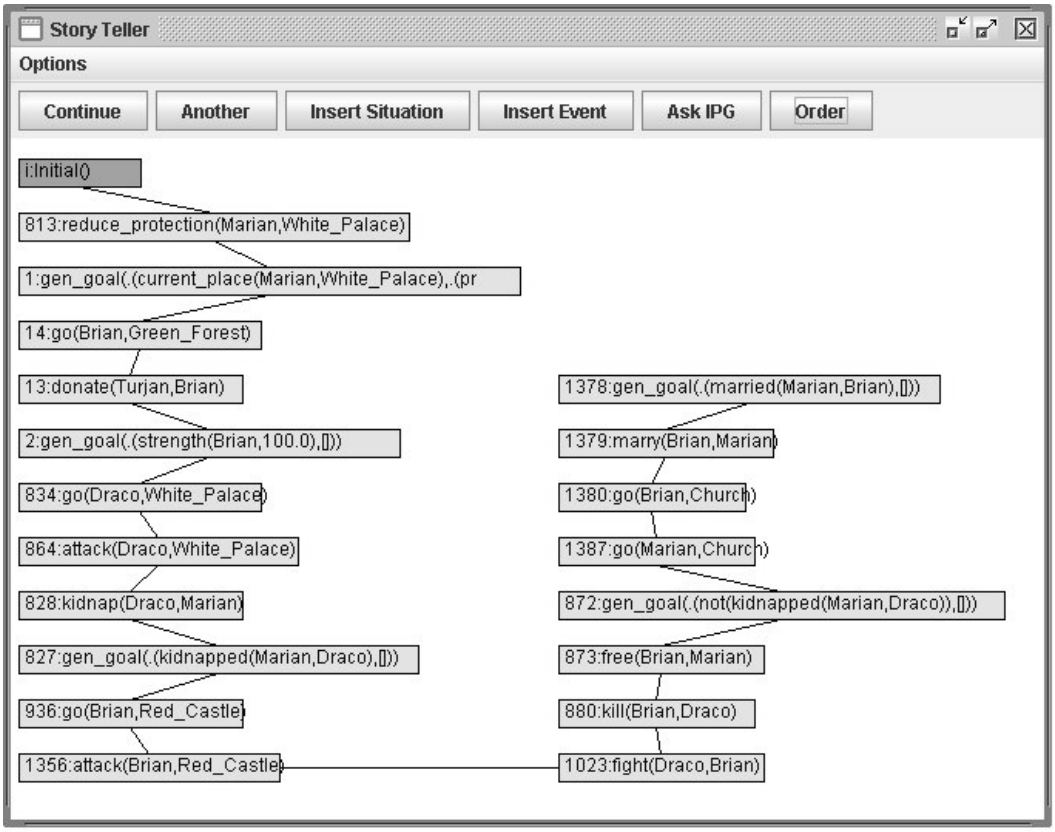

Figure 3: Example plot

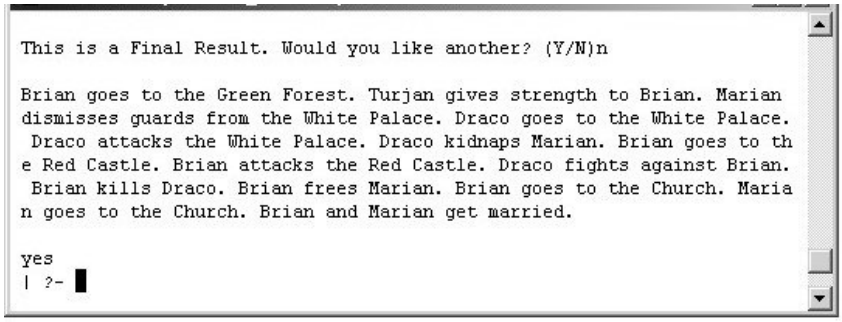

Figure 4: Template-based text

Figure 5: Draco attacks the White Palace

6. Concluding remarks

Following on Propp's footsteps, we proposed to extend his approach, originally restricted to fairy-tales, so as to be able to define literary genres in general. Contrasting with grammar-driven methods [33], predominantly concerned with the syntactical aspects of plots, our three-schemata conceptual modeling method is based on a plan-recognition / plan generation paradigm [21], covering the semantic and pragmatic aspects as well, since:

a. By feeding the mini-world factual description, provided by the static schema, into the definition of operations – expressing their pre-conditions and post-conditions in terms of such facts – we are able to determine the meaning of an entire plot, by simulating the successive state changes operated in the mini-world.
b. On the other hand, plots do not emerge by blind chance; they are set in motion by the goals of the participants. They can be regarded therefore as intentional sequences of actions, coherent with the different inclinations of the characters involved.

Such goals often exhibit a mutual dependence, determined by certain peculiar conventions of fictional genres. The assigned role largely determines what kind of conduct is expected from a given character, which in turn can only be deployed if the other characters also act as they are supposed to, always in accordance to their respective roles. Without this careful orchestration of goals, as we tried to achieve with the six goal-inference rules for our simple Swords and Dragons genre, the plots would fail to converge towards an appropriate outcome. Culler's insightful observation is helpful here [14, page 209]: "The plot is subject to teleological determination: certain things happen in order that the *récit* may develop as it does" – and he proceeds quoting Genette's allusion to the

"paradoxical logic of fiction", which requires that every unit of a story be defined by its functional qualities, among which are correlations with other units.

Although it seems adequate for characterizing genres where the stories exhibit a high degree of regularity, our proposal would not cope with the complexities of genres wherein the degree of variability is high. And, even for a genre that can still be treated, it would be presumptuous to claim that our specification would correspond exactly to the intuition of ordinary readers. We can define a genre G* merely as the set of plots P that our plan-based specification can recognize or generate. Surely we should try, as much as possible, to assess its closure with respect to the intended scope of the target genre G. Completeness proofs are in general harder than proofs of correctness.

An interdisciplinary approach, such as ours, opens promising perspectives. In particular, in this paper, we explored the combination of structuring constructs of literary origin (especially Propp's functions) with models familiar to computer scientists (such as the ER model, STRIPS, object and agent orientation, etc.). All features described here have been tested through the **LOGTELL** prototype tool. We are aware, however, that our modeling method, although minimally sufficient, should still be much enriched by a deeper study of modern literary theory and by incorporating contributions from other storytelling projects.

References

[1] Aarne, A. (1964) *The Types of the Folktale: A Classification and Bibliography*. Translated and enlarged by Thompson, S., FF Communications, 184, Helsinki: Suomalainen Tiedeakatemia, 1964.

[2] Afanas'ev, A. (1945) *Russian Fairy Tales*. N. Guterman (trans.), New York: Pantheon Books, 1945.

[3] Bal, M. (2002) *Narratology - Introduction to the Theory of Narrative*. University of Toronto Press, 2002.

[4] Bloom, H. (2003) *A Map of Misreading*. Oxford University Press, 2003.

[5] Borgida, A., and Brachman, R.J. (2003) “Conceptual Modeling with Description Logics.” In: Baader, F.; Calvanese, D.; McGuiness, D.L.; Nardi, D.; Patel-Schneider, P.F. (Eds) *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK.

[6] Calvanese, D.; De Giacomo, G. (2003) “Description Logics for Conceptual Data Modeling in UML”. In: Proc. 15th European Summer School in Logic Language and Information, August 2003, Vienna (Austria), pp. 18-29.

[7] Carlsson, M.; Widen, J. (1995) *SICSTUS Prolog Users Manual*, Release 3.0. Swedish Institute of Computer Science, 1995.

[8] Cavazza, M.; Charles, F.; Mead, S. (2002) “Character-based interactive storytelling”. IEEE Intelligent Systems, special issue on AI in Interactive Entertainment, 17(4):17-24, July 2002.

[9] Cervesato, I; Franceschet, M.; Montanari, A. (1997) “Modal Event Calculi with Preconditions”. In Proc. of the 4th. International Workshop on Temporal Representation and Reasoning, Daytona Beach, FL, USA, pages 38-45, May 1997.

[10] Ciarlini, A.E.M. (1999) *Geração Interativa de Enredos*. Ph.D thesis, Departamento de Informática, PUC-Rio, Brazil, 1999. (in Portuguese)

[11] Ciarlini, A.E.M.; Furtado, A.L. (1999) "Simulating the Interaction of Database Agents". In Proc. DEXA'99 Database and Expert Systems Applications Conference, Florence, Italy, Sept. 99.
[11] Ciarlini, A.E.M.; Furtado, A.L. (2002) "Understanding and Simulating Narratives in the Context of Information Systems". In Proc. ER'2002 – 21st. International Conference on Conceptual Modeling, Tampere, Finland, oct. 2002.
[12] Ciarlini, A.E.M.; Casanova, M.A.; Furtado, A.L.; Veloso, P.A.S. (2007) "Treating Literary Genres as Application Domains". Submitted for publication.
[13] Ciarlini, A.; Pozzer, C.; Furtado, A.; Feijó, B. (2005) A Logic-Based Tool for Interactive Generation and Dramatization of Stories, Dept. de Informática, Pontificia Universidade Católica do Rrio de Janeiro, Technical Report 07/2005.
[14] Culler, J. (1977) *Structuralist Poetics: Structuralism Linguistics and the Study of Literature*. London: Routledge & K. Paul, 1977.
[15] Elmasri, R.; Navathe, S. (2003) *Fundamentals of Database Systems*. Addison Wesley, 2003.
[16] Embley, D. (1997) *Object Database Development: Concepts and Principles*. Addison-Wesley, 1997.
[17] Fikes, R.E.; Nilsson, N.J. (1971) "STRIPS: A new approach to the application of theorem proving to problem solving", Artificial Intelligence , 2(3-4), 1971.
[18] Fillmore, C. (1968) "The Case for Case". In: Bach, E., Harms, R. (eds.), Universals in Linguistic Theory, Holt, Rinehart and Winston, 1968.
[19] Furtado, A.L. (1999) "Narratives and Temporal Databases: An Interdisciplinary Perspective". In: P.P.Chen, J.Akoka, H.Kangassalo, B.Thalheim (eds.), Conceptual Modeling: Current Issues and Future Directions, Springer-Verlag, 1999.
[20] Furtado, A.L.; Ciarlini, A.E.M. (1999) "Operational Characterization of Genre in Literary and Real-life Domains". In: Proc. of the ER'99 Conceptual Modeling Conference, Paris, France, November 1999.
[21] Furtado, A.L.; Ciarlini, A.E.M. (2000) "The Plan Recognition / Plan Generation Paradigm". In: Solvberg, A., Brinkkemper, S., Lindencrona, E. (eds.) Information Systems Engineering: State of the Art and Research Themes, Springer, 2000.
[22] Glassner, A. (2004) *Interactive Storytelling*. Natick: A K Peters, 2004.
[23] Goguen, J.A.; Thatcher, J.W.; Wagner, E.G. (1978) "An initial algebra approach to the specification, correctness and implementation of abstract data types". In: Yeh, R. T. (ed.), Current Trends in Programming Technology, Prentice-Hall, 1978.
[24] Huhns, M.; Stephens, L. (2000) "Multiagent Systems and Societies of Agents", In: G. Weiis (ed.), Multiagent Systems − a Modern Approach to Distributed Artificial Intelligence, MIT Press, 2000.
[25] Karlsson, B.; Ciarlini, A.E.M.; Feijó, B.; Furtado, A.L. (2006) "Applying a Plan-Recognition / Plan-Generation Paradigm to Interactive Storytelling". In: Proc. of Workshop on AI Planning for Computer Games and Synthetic Characters, ICAPS06, English Lake District, 2006.
[26] Kautz, H.A. (1991) "A Formal Theory of Plan Recognition and its Implementation". In: Allen, J. F. et al (eds.), Reasoning about Plans, San Mateo: Morgan Kaufmann, 1991.
[27] Koch, P. (1999) "Frame and Contiguity. On the Cognitive Basis of Metonymy and Certain Types of Word Formation". In: Panther, K. Radden, G. (eds.), Metonymy in Language and Thought, Amsterdam: John Benjamins, 1999.

[28] Kowalski, R.; Sergot, M. (1986) “A Logic-based Calculus of Events”. New Generation Computing, 4: 67-95, Ohmsha Ltd and Springer-Verlag, 1986.
[29] Miller, R.; Shanahan, M. (1994) “Narratives in the Situation Calculus”. Journal of Logic & Computation, Vol. 4, Number 5, 1994.
[30] Ozcan, R.; Aslandogan, Y. (2004) “Concept Based Information Access Using Ontologies and Latent Semantic Analysis”, Dept. of Computer Science and Engineering, University of Texas at Arlington, Technical Report 8, 2004.
[31] Propp,V. (1968) *Morphology of the Folktale*. Laurence, S. (trans.), Austin: University of Texas Press, 1968.
[32] Reiter, R. (1978) “On Closed World Databases”. In Logic and Databases, H. Gallaire and J. Minker (eds.), Plenum Press, 1978, pp. 55-76.
[33] Rumelhart, D.E. (1975) “Notes on a schema for stories”. In: Bobrow, D.G. and Collins, A.M. (eds) Representation and understanding: Studies in cognitive science, New York: Academic Press, 1975.
[34] Schank, R.C.; Abelson, R.P. (1977) *Scripts, Plans, Goals and Understanding*. Hillsdale: Erlbaum, 1977.
[35] Selden, R.; Widdowson, P. (1993) *A Reader's Guide to Contemporary Literary Theories*. The University Press of Kentucky, 1993.
[36] Sgouros, N.M. (1999) “Dynamic generation, managing and resolution of interactive plots”. Artificial Intelligence, 107, January 1999, pp. 29-62.
[37] Staab, S.; Studer, R. (eds.) (2004) *Handbook on Ontologies*, Springer, 2004.
[38] Velasquez, J.D. (1997) “Modeling emotions and other motivations in synthetic agents”. In: Proc. of the Fourteenth National Conference on Artificial Intelligence, Providence, 10-15, 1997.
[39] Willensky, R. (1983) *Planning and Understanding - a Computational Approach to Human Reasoning*. Addison-Wesley, 1983.
[40] Yang, Q.; Tenenberg, J.; Woods, S. (1996) “On the Implementation and Evaluation of Abtweak”. Computational Intelligence Journal, Vol. 12, Number 2, Blackwell Publishers 295-318, 1996.

Appendix I

```
/* STATIC SCHEMA */

entity(character,name).
entity(person,name).
entity(knight,name).
entity(princess,name).
entity(magician,name).
entity(dragon,name).
entity(place,place_name).

is_a(person,character).
is_a(knight,person).
is_a(princess,person).
is_a(magician,person).
is_a(dragon,character).

attribute(character,nature).
attribute(character,strength).
attribute(character,alive).
attribute(place,protection).

boolean(alive).
composite(protection,[kind,level]).

relationship(home,[character,place]).
relationship(current_place,[character,place]).
relationship(acquaintance,[character,character]).
relationship(married,[person,person]).
relationship(kidnapped,[person,character]).

attribute(acquaintance,affection).

role(hero,knight).
role(victim,(princess;knight)).
role(villain,(dragon;knight)).
role(donor,magician).
```

Appendix II

```
/* INITIAL STATE */

/* entity instances and their attributes */

db(knight('Brian')).
db(knight('Hoel')).
db(princess('Marian')).
db(magician('Turjan')).
db(dragon('Draco')).

db(nature('Brian',1.0)).
db(nature('Hoel',1.0)).
db(nature('Marian',1.0)).
db(nature('Draco',-1.0)).
db(nature('Turjan',0.0)).

db(strength('Brian',20.0)).
db(strength('Hoel',15.0)).
db(strength('Draco',45.0)).
db(strength('Marian',10.0)).
db(strength('Turjan',45.0)).

db(alive('Marian')).
db(alive('Brian')).
db(alive('Draco')).
db(alive('Hoel')).
db(alive('Turjan')).

db(place('White_Palace')).
db(place('Red_Castle')).
db(place('Gray_Castle')).
db(place('Green_Forest')).
db(place('Church')).

db(protection('White_Palace',[1.0,70.0])).
db(protection('Red_Castle',[-1.0,20.0])).
db(protection('Gray_Castle',[1.0,0.0])).
db(protection('Green_Forest',[0.0,20.0])).
db(protection('Church',[1.0,0.0])).

db(acquaintance([CH1,CH2])) :-
  db(character(CH1)), db(character(CH2)), dif(CH1,CH2).


/* relationship instances and their attributes */
/* note: not all values of the affection attribute are given */

db(home('Brian','Gray_Castle')).
db(home('Hoel','Gray_Castle')).
db(home('Marian','White_Palace')).
db(home('Draco','Red_Castle')).
db(home('Turjan','Green_Forest')).
```

```
db(current_place('Brian','Gray_Castle')).
db(current_place('Hoel','White_Palace')).
db(current_place('Marian','White_Palace')).
db(current_place('Draco','Red_Castle')).
db(current_place('Turjan','Green_Forest')).

db(affection(['Brian','Marian'],100.0)).
db(affection(['Hoel','Marian'],100.0)).
db(affection(['Marian','Brian'],0.0)).
db(affection(['Marian','Hoel'],0.0)).
db(affection(['Marian','Draco'],0.0)).
db(affection(['Turjan','Brian'],0.0)).
db(affection(['Turjan','Hoel'],0.0)).
db(affection(['Draco','Brian'],0.0)).
db(affection(['Draco','Hoel'],0.0)).
db(affection(['Brian','Draco'],0.0)).
db(affection(['Hoel','Draco'],0.0)).


/* Roles of the agents */

db(hero('Brian')).
db(hero('Hoel')).
db(victim('Marian')).
db(villain('Draco')).
db(donor('Turjan')).


/* a general ER rule */

db(X) :-
  \+ var(X),
  entity(E,_),
  X =.. [E,V],
  is_a(E1,E),
  Y =.. [E1,V],
  db(Y).
```

Appendix III

```
/* DYNAMIC SCHEMA */

operator_frame(1, go, [agent:(hero;victim;villain),destination:place]).
operator_frame(2, reduce_protection, [agent:victim,object:place]).
operator_frame(3, kidnap, [agent:villain,patient:victim]).
operator_frame(4, attack, [agent:(hero;villain;victim),object:place]).
operator_frame(5, fight, [agent:(hero;villain;victim),
  coagent:(hero;villain;victim)]).
operator_frame(6, kill, [agent:(hero;villain;victim),
  patient:(hero;villain;victim)]).
operator_frame(7, free, [agent:hero,patient:victim]).
operator_frame(8, marry, [agent:(hero;victim),coagent:(hero;victim)]).
operator_frame(9, donate, [agent:donor,recipient:hero]).
operator_frame(10, bewitch, [agent:donor,patient:(hero;victim)]).


operator(1,
      go(CH,PL1),
      [
      alive(CH),
          not(kidnapped(_,CH)),
          not(kidnapped(CH,_)),
          current_place(CH,PL0),
          dif(PL0,PL1)
      ],
      [
          not(current_place(CH,PL0)),
          current_place(CH,PL1)
      ],
      10,
      [current_place(CH,PL1)],
      [],[]) :-
          db(character(CH)),
          db(nature(CH,KIND)),
          dif(KIND,0.0),
          db(place(PL1)).

operator(2,
      reduce_protection(VIC,PL),
      [
         current_place(VIC,PL),
         protection(PL,[KIND,LPROT]),
         nature(VIC,KIND),
        { LPROT>0.0, LPROT1=LPROT-10.0 }
      ],
      [
        not(protection(PL,[KIND,LPROT])),
        protection(PL,[KIND,LPROT1])],
      10,
      [protection(PL,[KIND,LPROT1])],
      [],[]):-
          db(victim(VIC)),
          db(place(PL)).
```

```
operator(3,
      kidnap(VIL,VIC),
      [
           alive(VIC), alive(VIL),
           nature(VIC,KIND1),
           not(kidnapped(VIC,_)),
           strength(VIC,VIC_S),
           current_place(VIC,PL),
           protection(PL,[KIND2,LP]),
           strength(VIL,VIL_S),
           current_place(VIL,PL),
           dif(PL,PL1),
           {VIL_S>VIC_S+LP*KIND1*KIND2}
      ],
      [
            kidnapped(VIC,VIL),
            not(current_place(VIC,PL)),
            not(current_place(VIL,PL)),
            current_place(VIC,PL1),
            current_place(VIL,PL1)
      ],
      10,
      [kidnapped(VIC,VIL)],
      [],[]) :-
          db(victim(VIC)),
          db(villain(VIL)),
          db(home(VIL,PL1)).

operator(4,
        attack(CH,PL),
        [
         alive(CH),
         not(kidnapped(CH,_)),
         current_place(CH,PL),
         protection(PL,[KIND2,L_PROT]),
         dif(KIND1,KIND2),
          {
            L_PROT>0.0,
            L_PROT1 = L_PROT-30.0
          },
          affection([CH1,CH],La)
        ],
        [
          not(protection(PL,[KIND2,L_PROT])),
          protection(PL,[KIND2,L_PROT1]),
          not(affection([CH1,CH],La)),
          affection([CH1,CH],-100.0)
        ],
        10,
        [protection(PL,[KIND2,L_PROT1])],
        [],[]):-
        (
         db(hero(CH));
         db(villain(CH))
         ),
        db(nature(CH,KIND1)),
        db(place(PL)),
```

```
        db(home(CH1,PL)).

operator(5,
        fight(CH1,CH2),
        [
           alive(CH1), alive(CH2),
           nature(CH1,KIND1),
           nature(CH2,KIND2),
           dif(KIND1,KIND2),
           dif(KIND1,0.0), dif(KIND2,0.0),
           strength(CH1,LS1), strength(CH2,LS2),
           {
             LS1>=10.0, LS2>=10.0
           },
           current_place(CH2,PL), current_place(CH1,PL),
           protection(PL,[KIND3,L_PROT]),
           {
             L_PROT=<0.0,
             NEW_LS1=LS1-LS2,
             NEW_LS2=LS2-LS1
           }
        ],
        [
           not(strength(CH1,LS1)), not(strength(CH2,LS2)),
           strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)
        ],
        10,
        [strength(CH1,NEW_LS1), strength(CH2,NEW_LS2)],
        [],[]):-
        db(character(CH1)),
        db(character(CH2)).

operator(6,
      kill(CH1,CH2),
      [
       alive(CH1), alive(CH2),
       not(kidnapped(CH1,_)),
       nature(CH1, KIND1),
       nature(CH2, KIND2),
       dif(KIND1,KIND2),
       dif(KIND1,0.0), dif(KIND2,0.0),
       strength(CH1,LS1), strength(CH2,LS2),
       current_place(CH1,PL), current_place(CH2,PL),
       protection(PL,[KIND3,L_PROT]),
       {
        L_PROT*KIND3*KIND2=<0.0,
        LS2=<10.0, LS1>10.0
       }
      ],
      [not(alive(CH2))],
      10,
      [not(alive(CH2))],
      [],[]) :-
          db(character(CH1)),
          db(character(CH2)).
```

```
operator(7,
      free(HERO,VIC),
      [
        alive(HERO), alive(VIC),
        kidnapped(VIC,VIL), not(alive(VIL)),
        current_place(VIC,PL), current_place(HERO,PL),
       affection([VIC,HERO],LA)
      ],
      [
            not(kidnapped(VIC,VIL)), not(affection([VIC,HERO],LA)),
            affection([VIC,HERO],100.0)
         ],
      10,
      [not(kidnapped(VIC,VIL))],
      [],[]) :-
          db(hero(HERO)),
          db(victim(VIC)).

operator(8,
         marry(CH1,CH2),
      [
           alive(CH1), alive(CH2),
           affection([CH1,CH2],L1),
           {L1>80.0},
           affection([CH2,CH1],L2),
           {L2>80.0},
           current_place(CH1,'Church'),
           current_place(CH2,'Church'),
           not(married(CH1,_)),
           not(married(CH2,_))
      ],
      [
           married(CH1,CH2), married(CH2,CH1)
         ],
      10,
      [married(CH1,CH2), married(CH2,CH1)],
      [],[]) :-
          db(hero(CH1)),
          db(victim(CH2)).

operator(9,
         donate(CH1,CH2),
          [
            current_place(CH2,PL),
            alive(CH1),
            alive(CH2),
            affection([CH1,CH2],LA),
            strength(CH2,L1),
            {Alpha = max(0.0,min(1.0,LA+1.0))},
            {L2=Alpha*(L1+80.0)+(1.0-Alpha)*10.0}
          ],
         [
            not(strength(CH2,L1)),
            strength(CH2,L2)
          ],
      10,
      [strength(CH2,L2)],
```

```
      [],[]) :-
          db(donor(CH1)),
          db(home(CH1,PL)),
          db(hero(CH2)).

operator(10,
         bewitch(CH1,CH2),
      [
           nature(CH2,1.0),
           strength(CH2,LS),
           current_place(CH2,PL),
           alive(CH1),
           alive(CH2)
       ],
      [
           not(nature(CH2,1.0)),
           nature(CH2,-1.0),
           not(strength(CH2,LS)),
           strength(CH2,100.0)
           ],
      10,
      [nature(CH2,-1.0)],
      [],[]) :-
          db(donor(CH1)),
          db(home(CH1,PL)),
          db(character(CH2)).


/* templates for preparing legends */

template(go(CH,PL), [CH,' goes to the ',PL]).

template(reduce_protection(VIC,PL), [VIC,' dismisses guards from the
',PL]).

template(kidnap(VIL,VIC), [VIL,' kidnaps ',VIC]).

template(attack(CH,PL), [CH,' attacks the ',PL]).

template(fight(CH1,CH2), [CH1,' fights against ',CH2]).

template(kill(CH1,CH2), [CH1,' kills ',CH2]).

template(free(HERO,VIC), [HERO,' frees ',VIC]).

template(marry(CH1,CH2), [CH1,' and ',CH2,' get married']).

template(donate(CH1,CH2), [CH1,' gives strength to ',CH2]).

template(bewitch(CH1,CH2), [CH1,' bewitches ',CH2]).
```

Appendix IV

```
/* BEHAVIOURAL SCHEMA */


/* Goal-inference rules */


/* The strongest hero wants to become stronger
   than the villain */

rule(
     [
            e(i,strength(HERO,Lh)),
            e(i,villain(VIL)),
            e(i,strength(VIL,Lv)),
            h({Lh=<Lv})
     ],
     (
       [T],
      [
            h(T,strength(HERO,LS)),
            h({LS > Lv}),
            h(T>i)
     ],
     true
     )
  )
      :- findall(S,(db(strength(H,S)),db(hero(H))),Ss),
         max_list(Ss,Lh),
         db(hero(HERO)),
         db(strength(HERO,Lh)).


/* Victim spontaneously reduces the protection
   at her current location */

rule(
    [
         e(i,victim(VIC)),
         e(i,nature(VIC,KIND0)),
         e(i,current_place(VIC,PLACE)),
         e(i,protection(PLACE,[KIND1,PROT]))
     ],
     (
       [T],
        [
         h(T,current_place(VIC,PLACE1)),
         h(T,protection(PLACE1,[KIND2,PROT1])),
         h({(KIND2*KIND0*PROT1)<(KIND1*KIND0*PROT)}),
         h(T>i)
          ],
     true
     ) ).
```

```
/* If victim's protection is reduced, villain will
   want to kidnap her */

rule(
     [
         e(i,victim(VIC)),
         e(i,nature(VIC,KIND0)),
         e(i,current_place(VIC,PLACE1)),
         e(i,protection(PLACE1,[KIND1,PROT1])),
         e(i,villain(VIL)),
         h(g,alive(VIC)),
         h(g,current_place(VIC,PLACE2)),
         h(g,protection(PLACE2,[KIND2,PROT2])),
         h({(KIND2*KIND0*PROT2)<(KIND1*KIND0*PROT1)})
     ],
     (
       [T3],
      [
       h(T3,kidnapped(VIC,VIL))
      ],
      true
     )
   ).


/* If victim is kidnapped, hero will want to rescue her */

rule(
     [
        e(T1,kidnapped(VIC,VIL))
     ],
     (
       [T2],
      [
       h(T2,not(kidnapped(VIC,VIL))),
       h(T2>T1)
      ],
      true
     )
   ).


/* If victim is killed, hero will want to avenge her */

rule(
     [
         o(T1,kill(VIL,VIC)),
         h(T1,victim(VIC)),
         h(T1,villain(VIL))
     ],
     (
       [T2],
      [
       h(T2,not(alive(VIL))),
       h(T2>T1)
```

```
    ],
    true
   )
  ).


/* If the affection between two persons is high
   they will want to get married */


rule(
    [
       e(T,affection([CH1,CH2],L1)),
       h(T,affection([CH2,CH1],L2)),
       h(T,not(married(CH1,_))),
       h(T,not(married(CH2,_))),
       h({L2>95.0}), h({L1>95.0})
    ],
    (
     [T2],
     [
        h(T2,married(CH1,CH2)),
        h(T2>T)
     ],
     true
    )
  ).
```